Challenge 5

CY6740 – Machine Learning in CyberSecurity

Name: Tejas Krishna Reddy

NUID: 001423166

AWS Honey Pot Clustering Analysis

Solution:

1. Preprocessing:
    - We look at the missing values in each column. We observe that the maximum percentage of missing values in columns are seen in 'postalcode' and 'locale'. Now since we also use (latitude, longitude) along with another feature 'country', I thought it is better to remove 'postalcode' and 'locale' rather than impute them of any sort.
    - Now, since the number of samples with missing values are less than 10%, we now remove the rows with missing values.
    - There are a few instances, which have latitude > 20000 in them, which is a data entry error. And hence we remove the samples with latitude > 20,000.

Finally, we have 403,572 instances after removing the samples with errors or missing values.

- Now, we label encode the categorical columns. Namely – ['host', 'proto', 'srcstr', 'country'].
- Now, we apply kmeans clustering where k = [2, 3, 4, 5, 6]. We record all the Silhouette scores and check which k_value_cluster had the maximum score. **K=4 has best score.**

```
Silhouette Score for 2 clusters = 0.6925605989876972
Silhouette Score for 3 clusters = 0.684831669791793
Silhouette Score for 4 clusters = 0.7068276491145634
Silhouette Score for 5 clusters = 0.6770819867157432
Silhouette Score for 6 clusters = 0.6868073638815289
```

2. DBSCAN is applied to the same dataset with eps = 0.4 and min_samples = 10.
3. Both kmeans and DBscan clustering plots are shown as below for corresponding latitude and longitude.

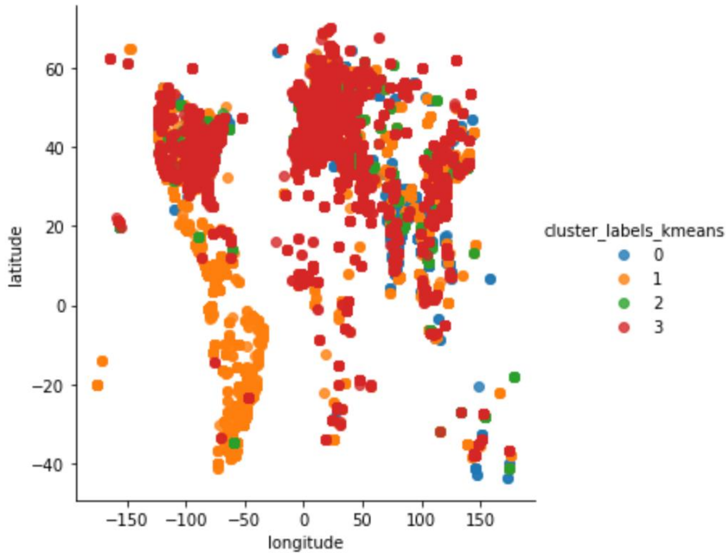


Fig 1: Kmeans (k=4) Clustering results, lat Vs lon

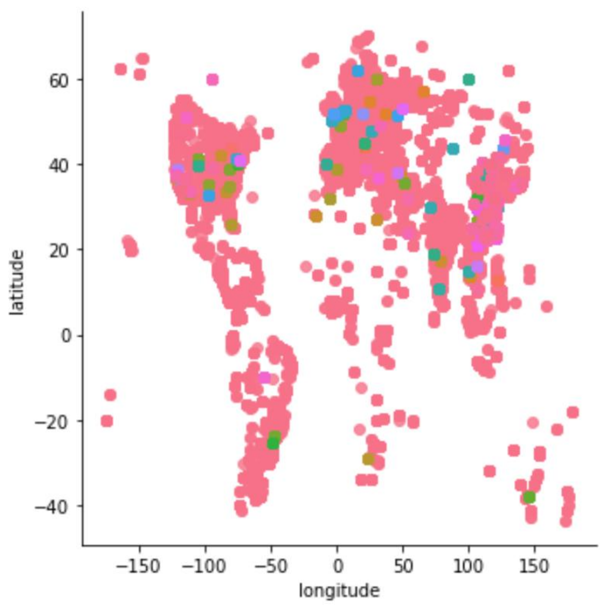


Fig 2: DBSCAN clustering analysis results. Lat Vs Lon

Challenge 5 - AWS Honey Pot Geo Clustering Analysis

- **Author: Tejas Krishna Reddy**
- **Date: 8th November 2020**

In [54]:

```
# Import packages
import pandas as pd
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
import seaborn as sns
```

In [55]:

```
# Read the dataset
df = pd.read_csv('AWS_honeypot_geo.csv')
df.head(5)
```

Out[55]:

	host	src	proto	spt	dpt	srcstr	country	locale	postalcod
0	groucho-oregon	1032051418	TCP	6000.0	1433.0	61.131.218.218	China	Jiangxi Sheng	Na
1	groucho-oregon	1347834426	UDP	5270.0	5060.0	80.86.82.58	Germany	NaN	Na
2	groucho-oregon	2947856490	TCP	2489.0	1080.0	175.180.184.106	Taiwan	Taipei	Na
3	groucho-us-east	841842716	UDP	43235.0	1900.0	50.45.128.28	United States	Oregon	9712
4	groucho-singapore	3587648279	TCP	56577.0	80.0	213.215.43.23	France	NaN	Na

Check how many null values exist in each column

In [56]:
```
df.isnull().sum()
```

Out[56]:
```
host                0
src                 0
proto               0
spt             44811
dpt             44811
srcstr              0
country          3634
locale         109469
postalcode     365103
latitude         3469
longitude        3428
dtype: int64
```

In [57]:
```
# % of missing in each column
df.isnull().sum() / df.shape[0] *100
```

Out[57]:
```
host           0.000000
src            0.000000
proto          0.000000
spt            9.923137
dpt            9.923137
srcstr         0.000000
country        0.804728
locale        24.241277
postalcode    80.849947
latitude       0.768190
longitude      0.759111
dtype: float64
```

Filtering:

- We have multiple columns that shows the location of the attack - like 'locale', 'postalcode', 'lattitude, longitude', 'country' etc. Hence, I am removing columns with high missing values such as 'postalcode' - which has more than 80% missing values in it and 'locale' which is again ~25% missing values.
- We also drop the rows with missing values after dropping these 2 columns.
- For a few records, latitude is mentioned above 20000 in value, which are a form of data error, and hence we remove those instances.

In [58]:
```
# Drop the columns with high missing values
df.drop(['postalcode', 'locale'],1, inplace = True)
```

In [59]:
```
# Now remove the rows with missing values
df.dropna(inplace = True)
```

In [60]:
```
# Remove the samples that have latitude value more than 20,000 (wrong- entr
df = df[df['latitude'] <= 18000]
```

In [61]:
```
### Print the number of instances that are remaining.
print("Total number of samples that remain after filtering: ", df.shape[0])
```

```
Total number of samples that remain after filtering:  403572
```

Convert Categorical variables to label encoded variables:

In [62]:
```
### Check the type of features each one is:
df.dtypes
```

Out[62]:
```
host          object
src            int64
proto         object
spt          float64
dpt          float64
srcstr        object
country       object
latitude     float64
longitude    float64
dtype: object
```

In [63]:
```
# Label Encode the categorical features:
le = LabelEncoder()
```

In [64]:
```
for cat_var in ['host', 'proto', 'srcstr', 'country']:
    df[cat_var] = le.fit_transform(df[cat_var])
```

Clustering Analysis:

- Do Silhoutte Analysis to find optimal number of clusters for kmeans.
- Do kmeans for the best_k, and store the labels.
- Do DBSCAN clustering and store the resultant variables.

In [65]:
```
df.head(1)
```

Out[65]:

	host	src	proto	spt	dpt	srcstr	country	latitude	longitude
0	2	1032051418	0	6000.0	1433.0	52428	36	28.55	115.9333

In [66]:
```
#df = df.reset_index(drop = True, inplace = True)
df1= df.sample(50000)

## Now apply kmeans clustering:
clusters = [2, 3, 4, 5, 6]

sil_scores = []
for c in clusters:
    cluster = KMeans(n_clusters=c, random_state=10)
    cluster_labels = cluster.fit_predict(df1)
    sil_scores.append(silhouette_score(df1, cluster_labels))
    print("Silhouette Score for {} clusters = {}".format(c,silhouette_score
```

```
Silhouette Score for 2 clusters = 0.6925605989876972
Silhouette Score for 3 clusters = 0.684831669791793
Silhouette Score for 4 clusters = 0.7068276491145634
Silhouette Score for 5 clusters = 0.6770819867157432
Silhouette Score for 6 clusters = 0.6868073638815289
```

In [67]:
```
# optimal k value with highest silhouette score is as follows:
best_ind = sil_scores.index(max(sil_scores))
best_k = clusters[best_ind]
print("Optimal number of clusters = ", best_k)
```

```
Optimal number of clusters =  4
```

In [68]:
```
# Redoing kmeans for best_k value:
cluster = KMeans(n_clusters=best_k, random_state=10)
cluster_labels_kmeans = cluster.fit_predict(df)
```

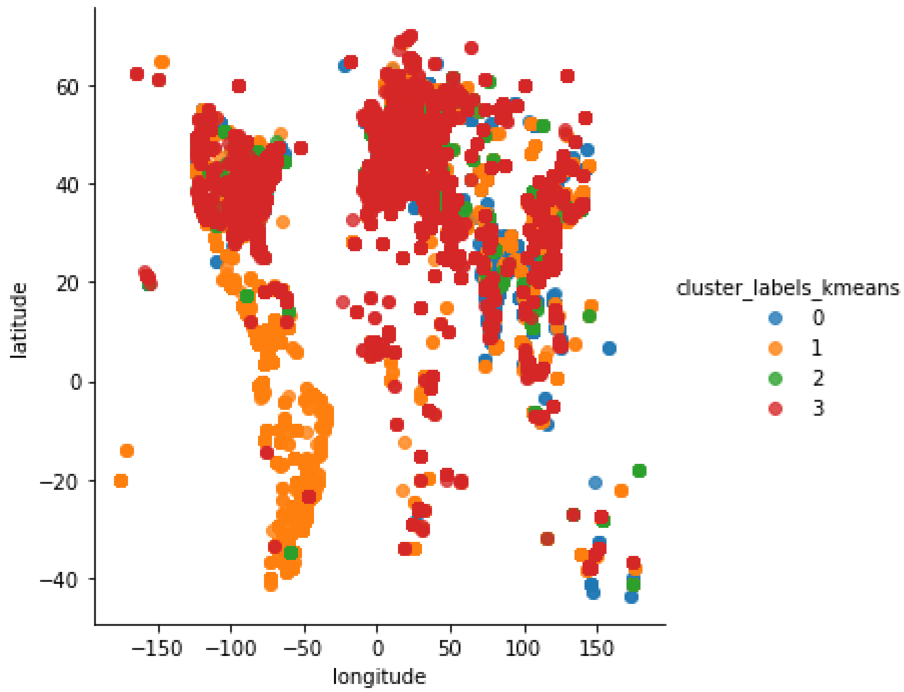
In [69]:
```
## Applying DBscan for the same dataset
cluster1 = DBSCAN(eps = 0.4, min_samples = 10)
cluster_labels_dbscan = cluster1.fit_predict(df)
```

Plot Scatter Plots with Latitude and Longitude with reference to Cluster Labels

In [70]:
```
## Plot labels with kmeans
df['cluster_labels_kmeans'] = cluster_labels_kmeans
df['cluster_labels_dbscan'] = cluster_labels_dbscan
```

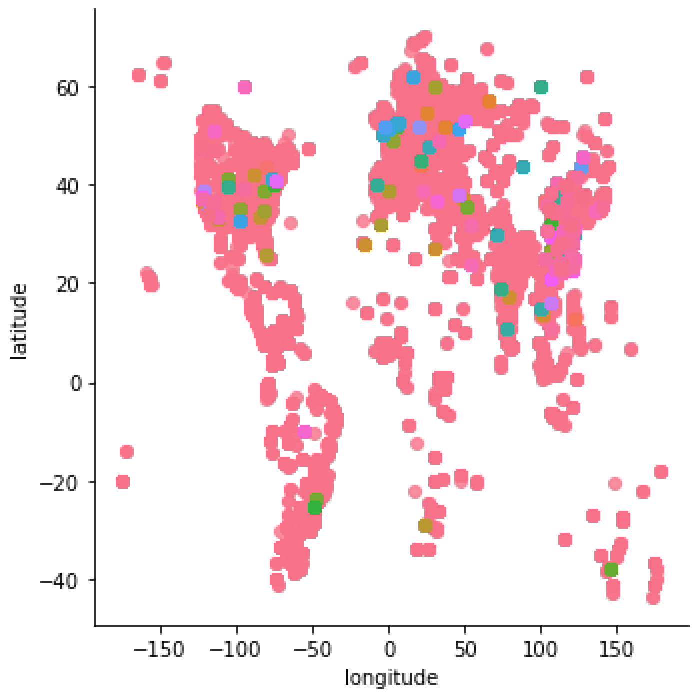
In [71]:
```
## Plot the weights Vs Volume with respect to each service line.
sns.lmplot(data=df, x='longitude', y='latitude', hue='cluster_labels_kmeans
```

Out[71]: <seaborn.axisgrid.FacetGrid at 0x1de4484a710>



In [72]:
```
## Plot the weights Vs Volume with respect to each service line.
sns.lmplot(data=df, x='longitude', y='latitude', hue='cluster_labels_dbscan
```

Out[72]: <seaborn.axisgrid.FacetGrid at 0x1df0b29ed68>



In [ ]:

In [ ]: